An Example of a Network-Based Approach to Data Access, Visualization, Interactive Analysis, and Distribution

Lee Elson, Mark Allen, Jeff Goldsmith, Martin Orton, and William Weibel
Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California

ABSTRACT

A freely available software package, called WebWinds, has been developed that allows atmospheric scientists, educators, students, and the general public to quickly and easily visualize and analyze data on many of the computer platforms in use today. WebWinds is written in Java and is able to ingest files from local disk or the World Wide Web (WWW). It is designed to eventually be distributed over the Internet and operate outside of WWW browsers entirely, allowing fewer restrictions as to where data and applications will be required to be stored. By manipulating data at their source, Internet bandwidth requirements will be reduced. The design and evolution of WebWinds, including its current display and analysis capabilities, security considerations, collaborative tools, availability, and planned future developments are described. Two typical sessions are also described to give the reader an introduction to its use.

1. Introduction and background

Computers have been used in the analysis of data, both meteorological observations and model output, for well over a generation. The recent explosive growth in the use of the Internet for the dissemination of observational data in both forecasting and research, together with new requirements for sharing observations and interpretations, has brought new challenges to tool developers and users. This challenge has been intensified by the rapid evolution of computer hardware, software, and network technology. Our group was originally charged with creating visualization and analysis interfaces between research scientists and data. We concentrated on the development of tools that allowed the display of large two- and three-dimensional datasets. During the past decade, our target audience has grown to include educators, students, and the general public around the world. Our approach has also changed from one focused on stand-alone client software on high-end Unix workstations to one in which multiple platforms, running different operating systems, must communicate over networks. This article describes our progress, including examples of analysis tasks using our freely available software package, and what we hope to achieve in the future.

There are two components to our approach. One involves the transfer of information among computers and among people, and the other involves techniques for understanding data through visualization. It is useful to provide a brief history of how our approach has evolved in both of these areas.

One way of categorizing visualization packages is to distinguish between those that use programming languages such as the Interactive Data Language, sold by Research Systems Inc.,[1] and those that use so-called visual approaches. The former allows the technically knowledgeable user a great deal of flexibility in the creation of an analysis or visualization package but requires the generation or modification of computer code. The latter creates a user interface that requires "point and click" input rather than programming and can be easier to use "out of the box." Adopting the second approach, we developed an application called

Corresponding author address: Dr. Lee Elson, M/S 183-501, Jet Propulsion Laboratory, 4800 Oak Grove Dr., Pasadena, CA 91109.
E-mail: elson@magus.jpl.nasa.gov
In final form 18 August 1999.

[1]Online at http://www.rsinc.com.

LinkWinds (Jacobson et al. 1994), which is distributed for free. As its name implies, LinkWinds allows the user to interactively link data, control and display components in Windows on the workstation screen, to produce a unique environment suitable for the task at hand. LinkWinds is designed to be intuitive to use, in part because it requires no knowledge of programming by the user. Many users can learn its capabilities by trial and error, referring occasionally to its integrated help system. By writing LinkWinds' code in the C programming language and using X-Windows for the graphical user interface (GUI), graphical display was optimized for speed and efficiency. This was crucial since at the time that LinkWinds was conceived, visualization requirements pushed the limits of workstation hardware, especially since we sought to provide the capability to handle large datasets.

In 1997, it became obvious to us that we needed a new approach to creating a package for visualization and analysis. Several factors contributed to this conclusion. Desktop computer processor speed and memory size had been increasing at a rapid rate. This was accompanied by an increase in the use of specialized graphics hardware and a decrease in prices, all of which contributed to a shift in the number and type of desktop computers in use in the scientific community. Many users have discovered that personal computers (PCs) running either Microsoft Windows or Apple's Macintosh Operating System (OS) provide a cost-effective platform for research-related activities. These users are generally not skilled at running applications under Unix operating systems. We rejected the idea of porting LinkWinds to PCs since we would have had to have devoted significant resources to the creation and maintenance of such ports at the expense of developing new functionality.

A second factor that influenced our conclusion was the growing popularity of the World Wide Web (WWW) for distributing data over the Internet. There are large, rich datasets currently available online, and projects such as the National Aeronautics and Space Administration's (NASA's) Earth Observing System promise a dramatic increase in the amount of available data in the next few years. As we discuss below, large datasets present both an opportunity and a challenge when one considers the limited bandwidth of the Internet and the limited capabilities of some desktop computers.

There are two additional trends in the analysis of scientific data made possible by the WWW. First, efforts are being made to make data interpretation understandable to the nonspecialist, including educators, students, planners, and the general public. Second, as the use of powerful desktop computers increases, it seems clear that an expanded audience will want to collaborate in their use of visualization tools.

2. New requirements, new approaches

Given these observations, we decided that there is a need for a freely available visualization and analysis package that inherits the performance and functionality of LinkWinds, that functions independently of computer platform and operating system, that allows users to collaborate over the Internet and that can be used as a stand-alone tool on a user's computer to analyze local data. We wanted a tool that could be used as a browser "helper application"[2] for downloading data from the WWW. Later development would allow tool components to be distributed over the Internet, independent of the WWW and browsers. We felt that Java (Campione and Walrath 1998; Java Quick Guide 1999), developed originally by Sun Microsystems, was most likely to fill these requirements.

Before creating such a package, we initiated a search for existing software that could meet the above requirements. Many capable packages are available but all are either limited to running on specific platforms, not freely available, require the user to use a programming language, have no collaborative capability, and/or serve only as client tools that cannot be distributed over a network. For example, the freely available client tool, Vis 5-D, is an excellent package, but cannot run on the Mac. The often used Grid Analysis and Display System package, also a client-only tool, does not allow collaboration. NCSA's[3] Horizon development kit, written in Java, is a collection of Java routines with 2D visualization capability. These routines must be assembled by the user, although some ready-made packages exist.

Because our concept for a visualization package differs from those available, we decided to proceed with our own development. Before making the decision to develop a package using Java, several issues

[2]A helper application is a program on the user's computer that is launched by a Web browser in order to provide a capability that is unavailable in the browser itself.

[3]National Center for Supercomputing Applications (http://www.ncsa.uiuc.edu).

that are important to developers and users needed to be addressed.

a. Performance: Network and programming considerations

There are several factors that affect performance. The execution time on the client machine is affected by the type of instructions that are generated by the program. Java is considered to be a platform-independent language. It does this by producing a type of code, called bytecode, that is interpreted in the same way on all Java-enabled platforms. This universality comes at a price however since the interpretation process is slow compared to the instructions (called machine code) generated by compilers such as those used for C or Fortran. Luckily, Java Just-In-Time compilers have been developed that significantly increase program execution speed. We have found that through careful programming, we can achieve performance that is very good for most[4] desktop platforms used today.

A more crucial issue is the transfer of information over the Internet. Originally, the WWW was developed with a simple operating premise: by constructing a unique address called a URL,[5] one could transfer a file from an Internet location to a user's browser. Later developments improved on this capability by allowing the user's browser to send the server a limited amount of information that a program on the server could use to create and return a unique file to the browser. Although these programs, called server-side CGI[6] scripts, greatly enhance the usefulness of the WWW, they still require that the server carry out all calculations in order to provide the user with a file.

In order to make efficient use of network bandwidth and of server resources, it is sometimes necessary to move some of the processing power from the server to the client. This has been a central concept in the evolution of Java (as well as other environments such as Virtual Reality Modeling Language). It has the ability to transfer an executable program, called an applet, from a WWW host to a client's browser. Although this enables a significant increase in functionality, a frequent observation of this type of WWW use is that it is slow. There are several reasons for this. First, before an applet can produce results, it must be transferred to the client's machine. Once there, the browser must load and execute the applet. Various browsers do this with differing efficiency since they have different standards for interpreting Java. These differences can result in execution errors. Applets also have limits to what they can do. For example, in many cases, security considerations prevent them from writing to a client's disk. Java addresses these (and other) shortcomings through the use of applications, or programs that function outside of a WWW browser, much like any other program on the client machine. Applications allow performance to be enhanced in several ways. In addition to the fact that they are not reliant on an inefficient WWW browser, if an application is to be used many times, it can be downloaded once, saving network bandwidth for other tasks like the transfer of data. Although there are disadvantages to using applications (e.g., setup requires user action outside of a browser), we have found that the advantages are significant, and, as a result, we have used them exclusively in our tool development activities.

Internet bandwidth also affects data transfer performance. The efficient transfer of data can be brought about by developing programs that only transmit data that are needed by the user. For example, many data archives create individual data files that contain broad spatial, temporal, or spectral coverages. Providing means to subset or subsample data at the source can greatly reduce bandwidth requirements. This is a primary design goal for our package and will be discussed further below.

b. Availability and standardization

As discussed above, Java bytecode may be executed on any Java-enabled platform. This is made possible by an interpreter, called a Java Virtual Machine (JVM). This kind of interoperability would be impossible unless developers of operating systems and applications adopt standards. Although Sun Microsystems has kept Java an open (i.e., nonproprietary) system, not all organizations involved with Java have agreed on a set of standards. At the present time, however, many manufacturers[7] have implemented JVMs that adhere to standards well enough to allow us to produce a working prototype visualization package. Sun has developed a JVM that runs under Microsoft Windows (95/98/NT) as well as one that runs under its own

[4]We recommend a processor speed of at least 100 MHz and at least 16 MB of memory.

[5]Uniform Resource Locator.

[6]Common Gateway Interface.

[7]Among these manufacturers are Apple, Silicon Graphics, IBM, and Hewlett-Packard.

Unix operating systems. Perhaps most important, these manufacturers have made their JVMs available for free. This is crucial for us since there would not be much point to our creating a freely available package if users were required to purchase the underlying interpreter. Although future actions with regard to standards are unpredictable, we are optimistic that there is a critical mass of cooperating manufacturers committed to the continued viability of Java.

c. Security and distributed computing

Security and functionality are often inversely related. In most cases that we have encountered, an analysis package needs to be able to at least write out new or modified data (including files containing snapshots of windows on the screen) to the local disk. Unlike applications, Java applets do not currently offer such capabilities. As we shall see, the full potential of a Java-based package will require the ability to transfer data and computational instructions from one platform to another. Security is a primary concern in the design and evolution of Java. For example, the language itself has many safety features that make it difficult to inadvertently (or maliciously) violate safety rules. Java verifies that bytecode has not been altered. Also, read/write/execute access restrictions can be configured easily in the latest version (1.2) of Java. Although these features cannot make an application risk free, they allow us to build a reasonable amount of security into our visualization package.

As mentioned above, Java was designed for network-based distributed computing. Although it is not unique in this regard, the standard Java package has the ability to invoke methods[8] from one platform while they reside on another platform or to bundle up a set of instructions and move them to another platform for execution. This very powerful feature will play a significant role in our future development efforts.

3. WebWinds: A Java visualization package

In designing our new Java visualization package, we felt that an evolutionary approach would work best. Since the development of LinkWinds gave us extensive experience in both what and what not to design into such a package, we began by building much of LinkWinds's functionality in Java. In order to highlight both the inheritance from LinkWinds and the new focus on the WWW, we named this new package WebWinds[9] (WebWinds 1999). To date, over 500 users have downloaded this package. Their interests span many disciplines including astronomy, planetary science, earth (including atmospheric and oceanic) science, and biology, and they represent individuals, schools, government labs, and the commercial sector.

During the initial development of WebWinds, we encountered difficulties due to incomplete implementations of Java on several platforms. This is to be expected when dealing with a new programming environment with ambitious goals, as is the case with Java. For this reason, and because we have extensive experience in building our own GUIs, much of WebWinds bypasses prebuilt Java components in favor of our own custom versions. This allows us greater control over both the "look and feel" and functionality of our package.

In order to maximize the utility of WebWinds as quickly as possible, we concentrated on four areas of development. First, we incorporated as much of the display and control functionality of LinkWinds as we could. Second, we built data ingestion tools and capabilities to make it as easy as possible to bring data into WebWinds. Third, we ensured that collaborative, Internet-based capabilities were present, and, fourth, we produced documentation and built packages that made it as easy as possible to download, install, and use WebWinds. The discussion that follows describes the capabilities of our version 1, released in August of 1999.

a. WebWinds applications suite

Space limitations prohibit a detailed discussion of WebWinds's application tools. Instead, the reader is referred to our online (and downloadable) documentation (Applications 1999). Here we will summarize these tools and provide examples in the appendix. As shown in the appendix (Table A1) there are three types of application tools. Most of the display tools are able to handle data with up to three dimensions (3D). Two dimensions are displayed while a third can often be changed. The displayed image is called a "slice." Display filters allow the user to select which two di-

[8]A procedure, like a subroutine, that performs a certain action.

[9]Online at http://webwinds.jpl.nasa.gov.

mensions are viewed. They also allow the user to select a portion of a large dataset for viewing. Control applications act to modify display applications. Examples of this are provided in the appendix.

As was the case with LinkWinds, the fundamental design feature of WebWinds is the ability to connect, or link together, both applications and other elements (collectively called "objects") such as those containing data. Each object occupies a window and windows can be connected using several different buttons. Operations that transfer data are connected through the use of a "drag and drop" button. In some cases, sequential operations are possible so that these objects can act first as a "consumer" of information and later as a "provider" of information that they may have modified. Control information, such as that provided by sliders, is exported via a "link" button. Examples of both types of connections are given in the appendix.

The reason for creating objects in windows and connecting them together is that the user is given flexibility in creating a session. For example, it is common to have one slider control two different image displays. Other types of controls (not listed in Table A1) are unique to a display, and are therefore enabled via a menu button on that display. For example, selecting "crosshair," "bounding box," or "grid" is only relevant to the display in which it is selected.

b. Data ingestion and output

In order to be useful, the user must be able to bring data into applications easily. This required us to build tools to read data files in a variety of formats. For simplicity, we have categorized files into two classes: self-describing and raw. Self-describing file formats, such as Network Common Data Format (NetCDF 1999), Hierarchical Data Format (HDF 1999), and HDF-EOS (HDFEos 1999) contain metadata[10] (e.g., Fig. A4a) as well as data and usually can be identified by a special "magic number" at the beginning of the file. Raw data, on the other hand, either contain no metadata or contain metadata that we do not read. Examples of this type of format include files containing only binary (e.g., floating point) numbers or "home grown" files where the creator has added nonstandard header information at the beginning. Another common example of a raw file format is ASCII, or textual data. All data files that have been compressed using the public domain gzip[11] compression utility are automatically uncompressed by WebWinds.

WebWinds's approach to the ingestion of data is to use a Data Wizard, or GUI-based input application, to do as much as possible automatically, providing a default configuration, and then to provide the user with several methods of adding to or modifying that configuration. Self-describing files often provide enough information for an initial analysis of a data file without any user input. Raw files require the user to specify such fundamental parameters as the number and size of data dimensions.

Successful data ingestion can involve more than just being able to read certain file formats. Depending on the capabilities of the user's machine and the size of the data file, it may be necessary to restrict the amount of data brought into computer memory. For example, an image with several thousand pixels on a side cannot be viewed on most computer screens without either reducing the resolution of the image or limiting the viewed area. WebWinds allows both of these options, which we call subsampling and subsetting, respectively. There are three ways in which this can occur. First, if the image is too large to even fit into memory (e.g., high-resolution, multispectral images), the user can either subset and/or subsample the data during the loading process. Currently the user does this by specifying (numerically or with sliders) the range and/or a skip parameter for each axis that is to be subset or subsampled. Soon, the user will be able to do visual subsetting: a resizeable rectangular box will be used to specify an area, in a display window, for subsetting. Second, if the image data will fit into memory, but not on the screen, the display tools will automatically subsample the image so that it fits onto the screen. Finally, the user can subset or subsample a data object after it has been read into memory using the "decimate/subset display" filter.

Data files used by WebWinds may be on the user's local disk or, as long as each file has a URL, anywhere on the WWW. As described below, files accessible from a WWW browser can also be imported into WebWinds. This means, for example, that a WWW site that creates data "on the fly" (e.g., a subsetting interface or database manager) can be used to provide input into the package.

[10]Data about data.

[11]Gzip is a Gnu compression utility that uses Lempel-Ziv LZ77 compression.

Saving results is often an important capability for an analysis package. Currently, a data object can be saved in several formats, including NetCDF and raw binary. In section 5, we discuss future enhancements in this area.

c. Download, installation, and documentation

WebWinds's software is freely available from our WWW page (WebWinds 1999), once the user has filled out a standard noncommercial license agreement. Although the bytecode is identical for each type of platform, we have created individual packages for Unix (SGI, Sun, and Linux), Windows (95/98/NT), and the Mac in order to facilitate installation. Packages for other platforms (e.g., OS/2) will be created soon. Each package contains the JVM (about 6 Mb), WebWinds software (3 Mb), documentation (5 Mb), and sample data (12 Mb), and totals about 26 Mb in size. Unpacking and setup are simple and straightforward for each package if one follows the instructions in the documentation.

If the user wishes to use WebWinds as a browser helper application, it is necessary to configure the browser for this capability, just as it is for most helper applications. The instructions for doing this vary with the platform and are included in the documentation. If both the browser and WWW server[12] are properly configured, clicking on a link will cause the associated file to be imported into WebWinds. Otherwise, the user must take the intermediate step of saving a file to his local disk before importing it into WebWinds.

WebWinds's documentation is written in HTML[13] and is available both on the WWW site and as part of the downloadable packages. Each application also has a "?" button that, when pressed, brings up a Java browser showing the documentation for that object. Thus, for example, if a user brings up a histogram application and does not know how to use it, pressing the "?" button will provide a description of that application. Included in the documentation package are several step-by-step examples of how to set up sessions. These sample sessions either use data included in the package or data available on the WWW. The descriptions include screen shots of the objects being described.

d. Automatic scripting: Rerun and remote sessions

By default, each WebWinds session generates a set of commands, called a script, that reflect the actions performed by the user. These commands are automatically saved in a file that must be renamed if a permanent version is desired. Any of these script files may be run during a WebWinds session and, with certain restrictions, may be run on other machines. Not only can a user easily recreate a session that was found to be useful, but such sessions can be given to collaborators for their own use. A data provider can set up a WWW site containing both data files and script files. Links to script files on a WWW site allow the user to import the scripts into WebWinds so that the process of setting up a session can be made automatic, once WebWinds has been installed. In addition, our development team can use script files to either help users with problems or investigate bugs in the WebWinds package.

The scripting language developed for WebWinds serves another purpose. It enables users to establish remote, shared sessions with other WebWinds users on the Internet. To do this, one user requests a shared session with another party by supplying an Internet address. If WebWinds is active at this address, a dialog box will open there asking for a connection. If the second user approves, all objects opened on the first user's desktop will open on the second user's desktop as well. The second user may also request a shared session with the first user, resulting in a collaboratory environment. In order for shared sessions to work properly, identical data files must be available to both users, either as duplicate files on their respective local disks or as a single file accessible to both on the Internet. Because only scripting commands are exchanged, these collaborative sessions can be carried out over low bandwidth connections.

4. Future work

By June of 2000, we plan to have in place several new or enhanced capabilities. Since saving results is important, we plan to add several file formats to our file save menu. We will accommodate other file formats, for both input and output, as the need arises. We also plan to add several new display and storage features. High on our priority list is a capability to save

[12]The WWW server must be configured to assign the correct mime type to a file. For example, an HDF file should have the type application/x-hdf. The user has little control over the server's configuration.

[13]HyperText Markup Language, the language used by the WWW.

image results as JPEG[14] or Postscript files suitable for publication. Also on the drawing board are several 3D displays (e.g., data on a globe or a transparent 3D image). Since Sun's 3D Java package, Java 3D, contains significant limitations and has not become a standard for 3D applications, we plan to use a standard graphics library, OpenGL. We also plan to enhance our animation capabilities. In the current version of WebWinds, it is possible to animate a display by using a slider. Future versions will include a tool that will allow the user to record and play back these animations. As other controls become available (e.g., a 3D rotator), they will be added to the animation toolkit. One other capability that we plan to add in the near term is the use of different map projections.

We also have plans to enhance the architectural options for WebWinds by the end of 2000. These enhancements will allow WebWinds to be used for data mining[15] activities. As we discussed in section 2, components of a Java application can be easily distributed over the Internet. This will allow us, for example, to move parts of WebWinds to a server or to several servers that can access datasets through local, high bandwidth connections. By doing this, the process of selecting only the data that is needed can be made much more efficient. Suppose data are organized at an archive so that very high spatial resolution global, atmospheric observations are stored in large files, each of which represent data taken during a 24-h period. If a user is interested in a time series of observations over a limited geographical region, most data archives today are structured so that it is necessary to obtain large amounts of data, most of which would be discarded. By placing subsetting software near the data source, the user would be able to avoid this inefficient transfer method, and would be able to access the data of interest much more quickly. We plan to develop server packages that would carry out these processes in the near future. We also plan to build interfaces to other database management systems, making it possible to carry out more sophisticated data queries and have the results fed directly into display and control applications.

Further in the future, it should be possible to move parts of WebWinds from one platform to another. Suppose there is a mirror site for a particular data archive. A "roving" version of WebWinds might assess the load level at several sites to determine which one would perform a function fastest, then move parts of a session to that site. Similarly, a user might enable an automatic update feature that would allow software updates or additions for his client machine to occur automatically, but only when needed. Finally, we intend to create "builder applications" that will enable user extensibility by allowing users to create their own applications and interfaces.

The amount of computing power connected to the Internet is enormous. Java offers a method for harnessing that power in a way that is invisible to the user, but highly beneficial to the scientific community.

Acknowledgments. The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Appendix: Sample sessions

In the appendix, we illustrate the capabilities of WebWinds by showing two examples of how it might be used. Because of space limitations, we can only present a very limited introduction to the use of the applications shown in Table A1. Our WWW site has several additional examples that give a more complete description of how to use the applications. Some of the screen shots shown below may differ from what the user will see due to changes in the released version of the software after the time of this writing.

Here, we offer two examples of how data might be brought into WebWinds and examined. Each example has a script file associated with it so that the user can create each session by simply selecting the appropriate script. For the first example, the script is included with the distribution package and can be selected under the "rerun" option of the "utilities" menu. The script describing the second example is available at our WWW site. It is highly recommended that the reader download and install WebWinds before reading the section below. Actually creating the sessions that we describe here will make the written description much easier to follow.

a. Example 1: Total Ozone Mapping Spectrometer (TOMS) monthly averages

The first example examines data from the Total Ozone Mapping Spectrometer instrument. These data represent monthly averages of total column ozone for 12 months in 1992 and 1993 and are contained in an

[14]Joint Photographic Experts Group.

[15]Data mining is the use of pattern recognition software tools to facilitate the acquisition of certain data from a data library.

TABLE A1. WebWinds application suite.

Name	Function
Displays	
Average	Averages slices of datasets and displays the results
Combine	Algebraically combines datasets and displays the results
Compare	Computes and displays vector statistics for a 3D dataset
Contour	Draws contours on several display applications
FFT	Displays the results of fast Fourier transforms
Histogram	Displays data distributions
Image	Displays data as 2D image
Line plot	*x*–*y* plot of data parallel to an axis
Profile	*x*–*y* plot of data along a specified line segment
Track pixel	Prints numerical values and statistics for a point or area
Value view	Displays numerical values for a slice of data
Window tool	Displays one dataset on top of another
2D scatterplot	Scatters one dataset against another (with statistics)
Display filters	
Decimate/subset	Creates a new dataset from a portion of an old one
View axis	Changes the viewing axes of a dataset
Controls	
Calculator	Allows algebraic manipulation of data for "combine"
Color tool	Choose/manipulate color palettes
Combine slider	Choose offsets for "combine" only
3-Slider	Choose offsets for "window tool" only
RGB slider	Choose three data slices as inputs to a color display
Slider	Pick a slice for any application to use

HDF file. There are two approaches that can be used to import these data. One approach would be to use the interactive Data Wizard to select the file and change or add to its metadata as is done in the second example, in the next section.

Another approach is to insert information about the observations into the WebWinds file, datamanager.txt. This file characterizes datasets used by the package and is organized into data directories, data source descriptions, file, and conversion types. Directories can be nested (one inside the other) so that they can contain other directories or data source descriptions in analogy with (but not directly related to) directories and files on a computer disk. Data source descriptions specify more than just the location of a source file on a disk. In addition to file name, path, and type, they contain information about the dependent and independent variables and associated color palettes. Table A2, modified slightly from the datamanager.txt file distributed with the package, contains a data source description listing that is used in this example. The first line assigns the name "92–93" to this source, which will appear in the desktop menu under "file" → "open." The second line specifies the name of the file to be used. Although not the case here, this can be a URL if appropriate. The third line indicates that the file is in HDF format. The fourth line instructs WebWinds to combine (concatenate) the separate monthly files into one 3D object. The next line is not really necessary but is included as a reminder that several simple mathematical conversions can be carried out as data from a file are loaded into the data object. The next four lines define the dependent and independent variables, giving them names, axis numbers, units, and value ranges. Note that we have

TABLE A2. Excerpt from datamanager.txt.

DataSource “92–93”
file “Ozone_Dec22_090631–0.hdf”
format “HDF”
concat
converter “”
meta “latitude” “degrees” axis 2 entries 180 range 90.0–90.0
meta “longitude” “degrees” axis 1 entries 360 range 359.0 0.0
meta “time” “months” axis 3 entries 12 range 1.0 12.0
meta “Ozone” “DU” axis 0 entries 254 range 0.324.
Colorhint “rgb”

changed the units in Table A2 to better reflect the associated variables and that the independent variable, here called “Ozone,” is always associated with “axis 0.” The last line assigns a preconstructed color table, called “rgb,” to the dataset.

Since the data source description just described is similar to that included in the WebWinds distribution, no additions to the distributed datamanager.txt are required, although one might change the units for consistency. WebWinds, when started, first produces a desktop window as shown in Fig. A1a. As described above, from the “file” menu, first select “open,” then select the “Earth” folder, then the “TOMS MONTHLY” folder, and scroll down to and select “92–93.” This brings up a data object that displays metadata read from the file and from datamanager.txt. Pressing the “load” button causes the data in the file to be read and produces the data object represented in Fig. A1b. Next we wish to display these data. We do this by first selecting “image” and then “slider” from the “tools” menu on the desktop. With the mouse cursor over the drag and drop button in the upper-right corner of the data object, press the mouse button and move the cursor to the image before releasing the button. Do the same with the slider. Finally, in order to tell the slider what it will control, connect the link button (the button with interlocking rings) on the slider to the image. The results are displayed in Figs. A1c and A1d. The image shows the distribution of column ozone averaged over a particular month. The month displayed can be changed using the slider. Notice that the image and slider both show, in numerical form, the month being displayed in the image.

There are several additional things to note about the image. One is the presence of a crosshair that can be moved using the mouse. The value of ozone under the crosshair is displayed just above the image itself. Another is that if one selects “coast180” from the “overlays” menu, a white outline of the coastlines of the world appears in the image. Although the script file for this example stops with the displays just discussed, there are several additional features that can be noted. Pressing the “line plot” button on the image produces

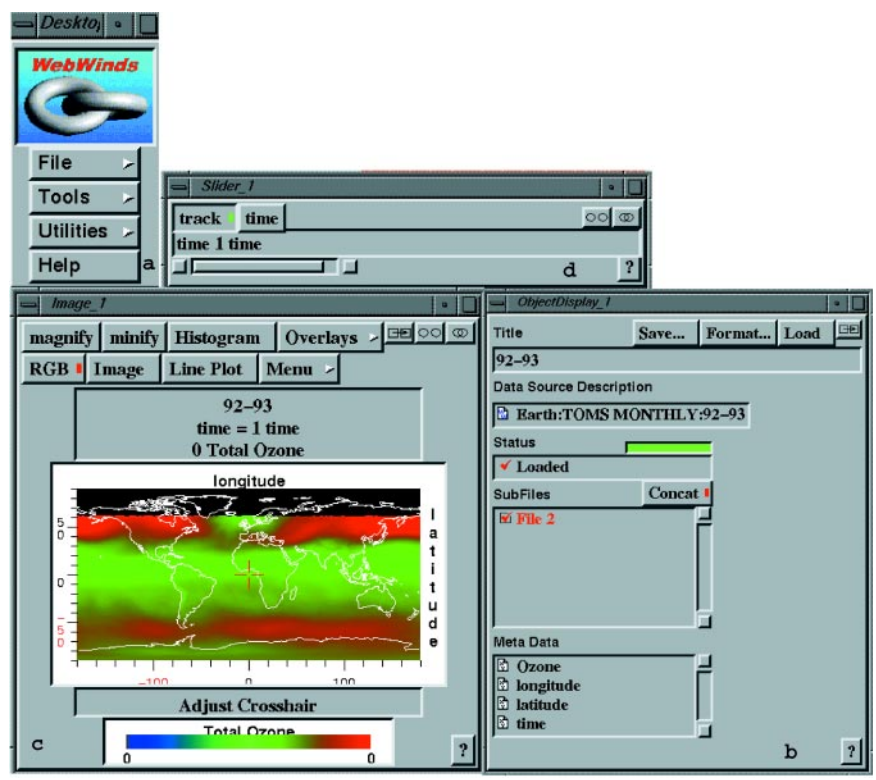

FIG. A1.(a) The desktop is the top-level window. (b) The object display for TOMS ozone. (c) The TOMS data in an image display, and (d) a slider used to control the time slice.

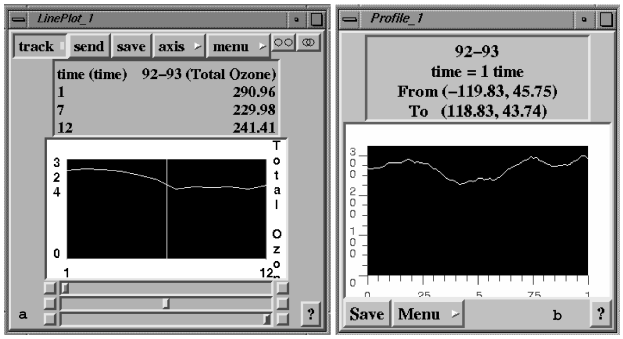

FIG. A2.(a) A line plot showing ozone (DU) vs time (months). (b) A profile showing ozone vs horizontal position.

an *x*–*y* plot for the point under the crosshair, as shown in Fig. A2a. For this example, this is a plot of column ozone as a function of time at the latitude and longitude of the crosshair. Notice a significant annual component to the variation in the line plot window (Fig. A2a). As the crosshair is moved, the line plot is updated. A similar tool, called profile, can be selected from the tools menu. This tool displays an *x*–*y* plot of data across an arbitrary path in the *x*–*y* plane of the image. First, use the drag and drop button to connect the data object to this new tool. Next, follow the directions in the profile window and link the image to the profile. In order to draw a profile on the image, select "Draw Profile Line"from the "profile line" option on the image menu button. Then use the mouse to actually draw the line on the image.

b. Example 2: Climatology Interdisciplinary Data Collection

The second example uses data from the Goddard Distributed Active Archive Center (DAAC).[16] Our intent is to demonstrate that any file accessible on the WWW can be treated as if it were a local file, apart from the latency inherent in the process of downloading the data. We believe that this has relevance to data producers, providers, and users.

We have chosen to use products from the Climatology Interdisciplinary Data Collection[17] because of the availability of data representing observations of several atmospheric and surface parameters that have been mapped to a common grid. It is important to note that data need not be on a common grid to be compared in WebWinds. This example differs from the previous one in that the data are not self-describing; that is, the files contain no metadata. Instead, each file holds data in floating point format and the user must read the documentation to know how to interpret the numbers in the file.

To begin this session, we start WebWinds as in the previous example, obtaining the desktop window (Fig. A1a). Next we select "new source" from the "files" menu, obtaining a window similar to that in Fig. A3a. Here, we must enter the URL[18] for the first data file we wish to use in the text pad marked "current file." We have selected surface temperature data derived from the TIROS Operational Vertical Sounder instrument. With the file selected, press the "open" button to get an object display window. Because this file is not recognized, the user must select a file type from the "file type" menu button. The appropriate value here is "raw float." When this selection is made, several other required fields appear in the window, as shown in Fig. A3b. Since the data are two-dimensional, the number 2 must be entered in the "number of dimensions" text area and "360 180," indicating that the data represent a 360 × 180 (longitude × latitude) array, must be entered in the "size of dimensions" area. Finally, a title should be added to the "title" area. We have chosen "surface temperature." With these specifications complete, pressing the "open" button checks to see whether the user has entered illegal choices. If not, another press of the "open" button causes the window to display a summary of the data attributes, including default metadata (in the "meta data" window) describing the dependent and independent variables. At this point, no data have been read from the file. Pressing the "load" button does this, providing information about the data values in the file, as shown in Fig. A3c. Notice that the metadata window shows default names for the variables. In order to change these names, as well as the units and data ranges, the user must click on each of the metadata lines in turn. For example, clicking on the first one, titled "value," brings up a separate window. Selecting "edit" from the menu button in this new window allows the independent variable to be given the title "temperature," as illustrated in Fig. A4a. Notice that the "start value" was changed from –999 to 200. This was done to eliminate the "no data" values in the file. In order to make

[16]For more information on the DAAC visit http://daac.gsfc.nasa.gov/DAAC_DOCS/gdaac_home.html online.

[17]Online at http://daac.gsfc.nasa.gov/CAMPAIGN_DOCS/FTP_SITE/inter_disc.html.

[18]Available online at ftp://daac.gsfc.nasa.gov/data/inter_disc/tovs_atmo_sound/tsurf/1988/tovsng.tsurf.1pmegg.8801.bin.

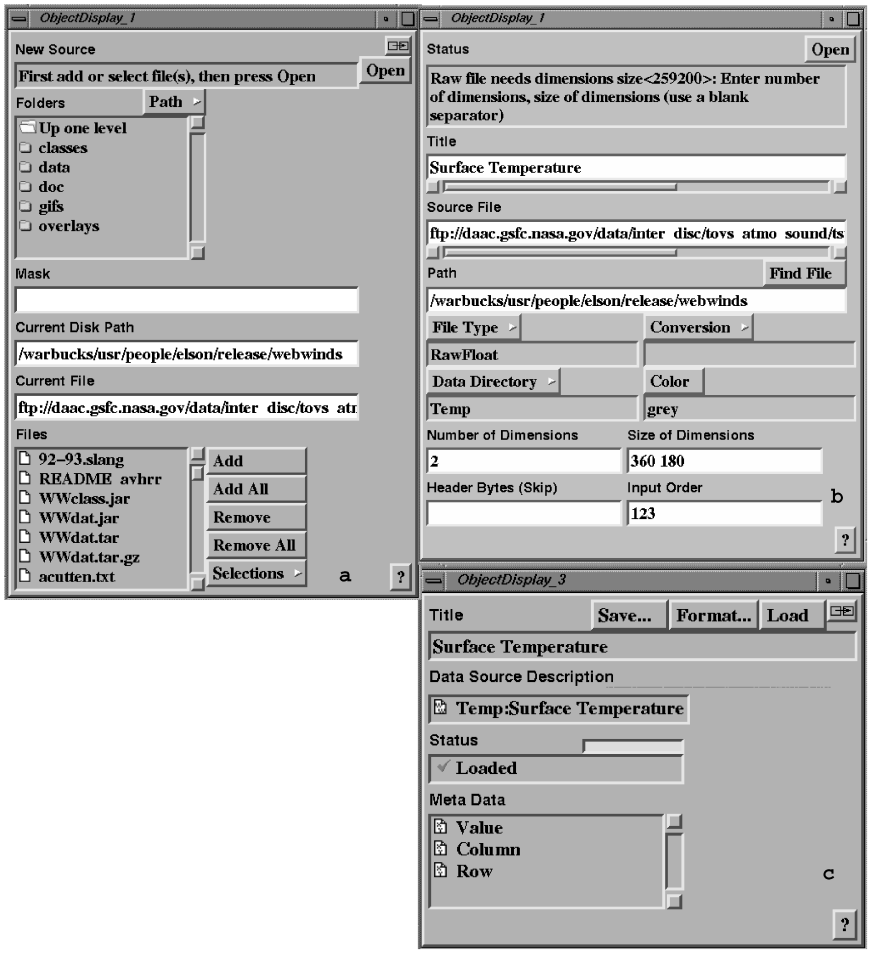

FIG. A3. The object display used to (a) locate a file, (b) specify file parameters, and (c) represent a loaded data object.

this effective, the "limit data" control flag must be checked. To make these metadata changes effective, "apply" must be selected from the menu. This window can then be closed. The two dependent variables, longitude (–180° to 180°) and latitude (–90° to 90°) (called "row" and "column" in Fig. A3c) can also be edited using the same approach. When all metadata have been modified, the "load" button on the object display must be pressed.

With one data object created, it is time to choose a second for comparison. We have selected outgoing longwave radiation (OLR) data derived from the Earth Radiation Budget Experiment. Again, starting with the "new source" option from the "files" menu, we can create an OLR data object. The URL[19] will be different, of course, as will the dependent data variable as specified in the metadata window. Be sure to press "load" on this new data object, once all modifications have been made to metadata.

With two data objects created, we return to the desktop and select "image" from the "tools" menu. Use the drag and drop button in the upper-right corner of the surface temperature object display to connect the temperature to the image. Next, select "2D scatterplot" from the same "tools" menu and first drag the temperature and then the OLR objects into it. As shown in Fig. A5a, the resulting display shows a scatterplot, as well as some simple statistics for the two datasets. There appear to be two distinct correlative regimes between OLR and surface temperature. Notice that the scatter diagram is color coded. The colors correspond to different loca-

[19]Available online at ftp://daac.gsfc.nasa.gov/data/inter_disc/radiation_clouds/erbe_rad/1988/erbe.lwolr.lnmegg.8801.bin.

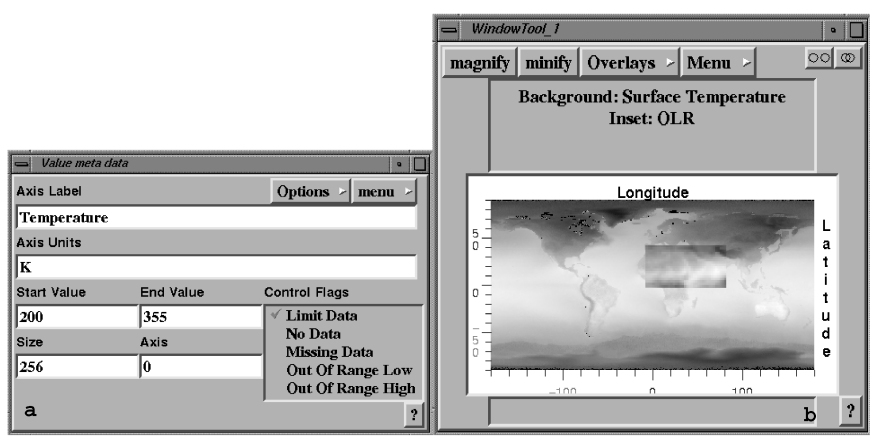

FIG. A4.(a) An example of a metadata object showing labels, units, and values. (b) A window tool object containing an image showing surface temperature in the background and OLR in the small inset window.

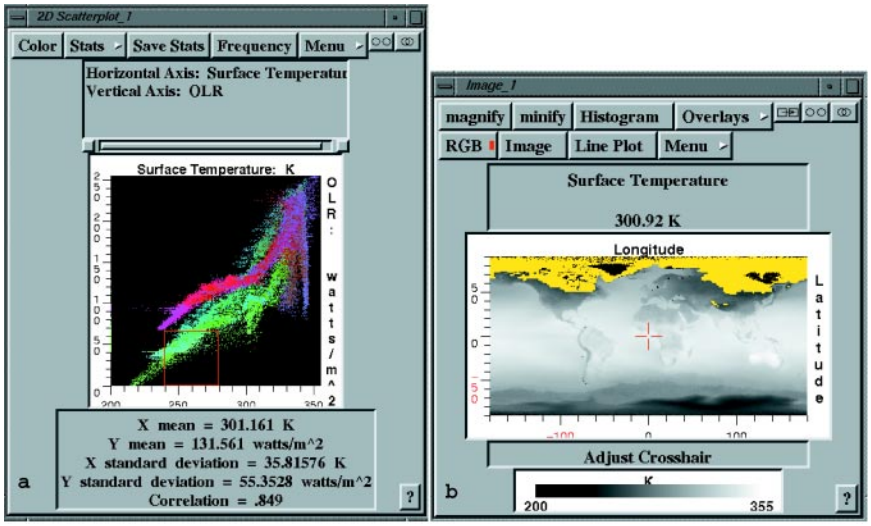

FIG. A5.(a) A 2D scatterplot showing the relationship between surface temperature and OLR.The scatter points are color coded, representing different spatial locations. (b) An image showing the surface temperature data. The yellow area shows the location of the data inside the red bounding box in the 2D scatterplot.

tions on the earth's surface. A more precise way to determine the geophysical location of points in the scatterplot is to select "resize bounding box" from the "bounding box" submenu of the "menu" button on the scatterplot. As shown in Fig. A5a, this allows a box to be drawn over a portion of the data in the scatter diagram. If we now use the "link" button (top-right button) in the scatterplot to connect this display to the image, we see the data points inside the box are highlighted in yellow in the image window (Fig. A5b).

WebWinds has a second tool that is useful for comparing two datasets. From the tools menu, select "window tool." As with the scatterplot, first drag the surface temperature object display and then the OLR object display into this tool. The result, shown in Fig. A4b, is a display with temperature in the background and OLR in a small inset window. The inset can be resized and moved over the background image.

References

Applications, cited 1999: WebWinds Applications suite. [Available online at http://webwinds.jpl.nasa.gov/rel1/webpage/webhelp.html.]

Campione, M., and K. Walrath, 1998: *The Java Tutorial.* Addison-Wesley, 964 pp. [Available online at http://java.sun.com/docs/books/tutorial/index.html.]

HDFEos, cited 1999: HDF-EOS. [Available online at http://spsosun.gsfc.nasa.gov/InfoArch_docs.html.]

HDF, cited 1999: Hierarchical Data Format. [Available online at http://hdf.ncsa.uiuc.edu.]

Jacobson, A. S., A. L. Berkin, and M. N. Orton, 1994: LinkWinds: Interactive scientific data analysis and visualization. *Commun. ACM,* **34,** 43–52.

Java Quick Guide, cited 1999: A quick guide to the Java platform. [Available online at http://java.sun.com/nav/whatis/index.html.]

NetCDF, cited 1999: Network Common Data Format. [Available online at http://www.unidata.ucar.edu/packages/netcdf/.]

WebWinds, cited 1999: WebWinds documentation and software. [Available online at http://webwinds.jpl.nasa.gov.]